Toward FPGA-Based Semantic Caching for Accelerating Data Analysis with Spark and HDFS

Marouan Maghzaoui, Laurent d'Orazio, Julien Lallet

Toward FPGA-Based Semantic Caching for Accelerating Data Analysis with Spark and HDFS

Marouan Maghzaoui[1], Laurent d'Orazio[2], Julien Lallet[1]

[1] Nokia Bell Labs {marouan.maghzaoui,julien.lallet}@nokia-bell-labs.com
[2] Univ Rennes, CNRS, IRISA laurent.dorazio@irisa.fr

Abstract. With the increase of data, traditional methods of data processing have become time and power inefficient. As enhancement, we propose a new accelerated architecture for querying big Databases. This architecture combines the advantages of the HDFS for the management of huge amount of data and the fast processing of queries of Spark SQL. It also benefits of the processing efficiency of the hardware acceleration of FPGAs and of the semantic caching architecture to process recently used data stored in the cache.

Keywords: FPGA, Spark, HDFS, Semantic Caching.

1 Introduction

One of the big challenges in the IT sector today is the handling of an ever-growing size of databases. The annual size of data that is created and stored worldwide increases by 31% every year and amount of database computations increases by 21% every year [6].

Processing and analyzing all this data is a big challenge. It requires a mix of processing techniques, data sources and storage formats. Query executions and computations must be accelerated by large rate. Furthermore, large storages must be handled efficiently to manage many read and write operations. This issue is raised in many applications: the Internet, social network, healthcare, cyber security, smart cities, etc. In particular, security monitoring, a domain of cyber security aims to store large amount of logs so as to detect subtile attacks. This require on one hand large storage and on the other hand efficient processing.

Until now, Generic Purpose Processors (GPPs) are used to compute database queries. But GPPs now face the near end of Moore's Law [16], limits in terms of clock frequency [13] and multi-core processing [9]. Hence, we must think of further possibilities to accelerate the query executions.

Recent works [1], [2], [12] have shown that we could use distributed file systems (DFS) and Massively Parallel Processing (MPP) to store and process huge databases and complex queries. Hadoop Distributed File System (HDFS), the open source version of the google file system GFS [10], and Spark SQL [2] are respectively popular DFS and MPP used by the big data communities. HDFS and Spark rely on GPPs and consequently inherit from GPP limitations mentioned

previously. Regarding hardware acceleration, [4], [8], [14], [15], [18] have shown that Field Programmable Gate Arrays (FPGAs) could be a better alternative to GPPs in the field of database applications in general and for query-execution purposes with an increasing data throughput with lower energy effort. But not all queries can be easily implemented and processed in a FPGA. It is necessary to have a standard SQL engine to run those non-supported queries. Furthermore, it is reasonable to consider that DFS should be still managed by software. Anyway, intern memory cells of the FPGAs and dedicated RAM memories on board should be used as cache memories to store recent data and results. Semantic caching, which is a technique used for optimizing the evaluation of database queries by caching results of old queries and using them when answering new ones, shows a lot of promise and can be applied to accelerate database queries [17].

This paper describes our effort to create an architecture that combines an accelerated query engine and a semantic cache memory implemented on FPGA, HDFS and Spark to accelerate query executions. This architecture benefits from the hardware acceleration of FPGA, the rapid access to memory in the semantic cache and Big Data capabilities of HDFS and Spark to efficiently execute very large queries.

The remainder of the article is outlined as follows: Section 2 discusses background and related work in the field. Section 3 introduces the proposed architecture to accelerate query processing. Finally, a conclusion and some perspectives are given in the last section.

2 Background

This section provides necessary background on Big Data, hardware acceleration and semantic caching. Later, we present recent related research and we discuss their limitations.

2.1 Big Data

According to [5], the term *Big Data* is used to refer to the huge volume of data that are too difficult to store, process and analyze through traditional database technologies like CPU servers in conventional Data Centers within a tolerable time. Some of the solutions used by the Big Data and database communities to handle the problems previously mentioned are Spark SQL and HDFS.

HDFS is a self-healing, distributed file system that provides reliable, scalable and fault tolerant data storage on commodity hardware. HDFS accepts data in any file format like text, images, videos, database regardless of architecture and automatically optimizes high bandwidth streaming. HDFS is designed to hold large amounts of data and provides faster access to data. It is highly scalable anyway limited to 200 Peta Bytes (PB) of storage [3].

Spark is a general-purpose cluster computing engine with libraries for streaming, graph processing, machine learning and SQL. Spark SQL provides a Data

Frame API that can perform relational operations on both external data sources and Spark's built-in distributed collections. This API evaluates operations lazily so that it can perform relational optimizations. Also, to support the wide range of data sources and algorithms in Big Data, Spark SQL introduces a novel extensible optimizer [2]. Spark SQL is also a part of Hadoop Ecosystem and can be deployed directly on HDFS so that resources can be statically allocated on some of the machines of an Hadoop cluster.

2.2 Hardware Acceleration

As we reached the end of Moore's law [16], we are experiencing a growing interest in new solutions to boost performance.

The combination of different hardware accelerators, as Graphic Processing Units (GPUs), FPGAs and Application Specific Integrated Circuits (ASICs) along with GPPs, provides a wide panel of solutions from which to choose the most suitable architecture for a specific task. Among these architectures, FPGA provides an excellent acceleration platform. Inherent parallelism, access to local memories as Block Random Access Memories (BRAMs) and the ability to be partially and dynamically reconfigured are interesting characteristics we need to efficiently exploit.

2.3 Semantic Caching

Caching consists in a resource element to accelerate computing processes. Data stored in a cache might be the result of an previous request or a duplicate of recently used data stored elsewhere. Traditional cache architectures are based on page and tuple caching where possible data relationships are not handled efficiently. Semantic caching has been proposed to overcome these drawbacks.

The cache is managed as a collection of semantic regions which are groups of semantically related data. Access and cache replacement are managed at a unit of semantic region. Semantic caching is based on three key features. First, a description of the data stored in the cache is maintained in the form of a compact specification. Requests for missing data in the cache are thus faster fetching to the cache. Secondly, replacement policies are flexible and could be different for each semantic regions, which are associated with collections of tuples. This is to avoid the high overheads of tuple caching and, unlike page caching, is insensitive to bad clustering. Third, maintaining a semantic description of cached data enables the use of sophisticated value functions that incorporate semantic notions of locality, not just Least Recently Used (LRU) or Most Recently Used (MRU) policies in case of cache replacement [7].

Queries in a semantic cache are split in two parts: a remainder and a probe query. The probe query retrieves the portion of the result already available in the cache. A remainder query fetches any missing data in the cache.

For example, figure 1 represents a semantic cache storing log data. The semantic regions are arranged according to the time and IP addresses of the logs.

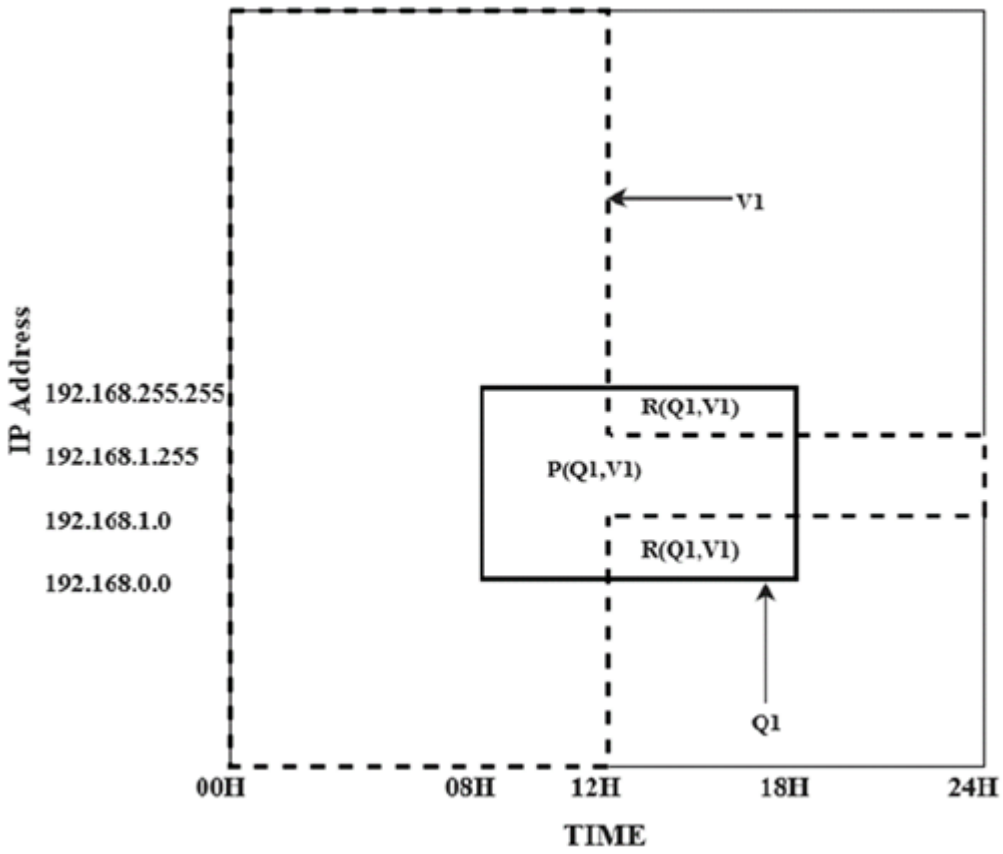

Fig. 1. Example of semantic cache regions

V1 (equation 1) represents the result of previous queries (Q_{n-1} (equation 2) and Q_{n-2} (equation 3) already stored in the cache.

$$V1 = (Time < 12H \vee (IPAddress > 192.168.1.0 \wedge IPAddress < 192.168.1.255)) \tag{1}$$

$$Q_{n-1} = (Time < 12H) \tag{2}$$

$$Q_{n-2} = (IPAddress > 192.168.1.0 \wedge IPAddress < 192.168.1.255) \tag{3}$$

$$Q1 = ((Time < 8H \wedge Time > 18H) \wedge \\ (IPAddress > 192.168.0.0 \wedge IPAddress < 192.168.255.255)) \quad (4)$$

A new query Q1 (equation 4) would be compared to V1 (V1∧ Q1) to fetch the probe query in the cache (P (Q1, V1) as depicted figure 1). The remainder query (R (Q1, V1)) is then executed to get the missing data. The results of remainder and the probe queries are combined and sent to the user. The result of the remainder query is stored in the cache for future queries.

2.4 Motivation

Static approaches which use FPGAs for query processing [4], [8], [18] succeeded in implementing query engines capable of doing SQL based processing (select, project join, etc.) with a very high throughput. Some recent research proposes a hybrid architecture (FPGA + CPU) to process LIKE queries [14].

Although some of these existing solutions handled queries for large volumes of data (100 TB in [18]), to the best of our knowledge, the proposed FPGA architectures cannot process huge data stored in multiple servers by the use of a distributed file system. The architecture that we propose, using the Distributed File Systems HDFS, massively parallel processing on Spark SQL, hardware acceleration and semantic caching in FPGA could be the solution to the problem of executing queries in Big Data in a relatively short time.

3 Toward FPGA-Based Semantic Caching for Accelerating Big Data Analysis with Spark and HDFS

In this section, we describe the preliminary building blocks for a FPGA-based semantic caching for Big Data analysis's architecture. First, details of our hardware and software architecture is presented followed by some description of the SQL queries execution. To start with, the system will consider SQL queries with conjunctions only.

In a second part, the FPGA architecture implementation is given. In the following, we detail the management of semantic regions, in particular reads and writes in the FPGA. Analysis and evaluation on the cache entries will be addressed in future works and are out of the scope of this paper.

3.1 Overview

The FPGA-based semantic caching for Big Data analysis's architecture that we propose (figure 2) is a hardware software implementation system with a PCI Express interface for inter communication.

The FPGA contains the semantic cache and an accelerated query engine. The software part contains the Query analyzer and Spark. HDFS is the file system where the database is stored.

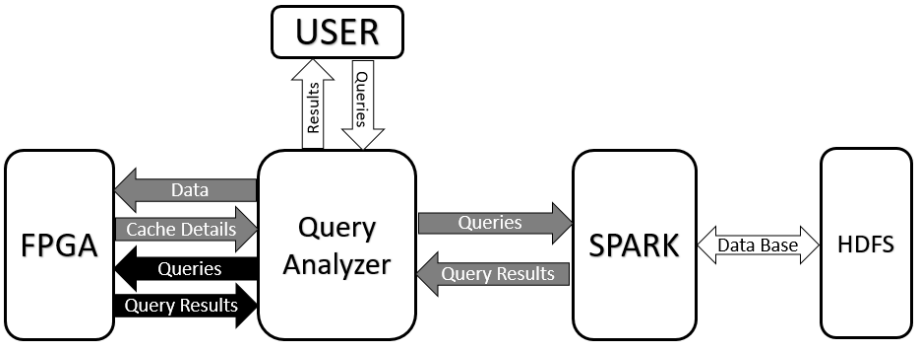

Fig. 2. Architecture of FPGA-based semantic caching for Big Data analysis

The Query analyzer contains a catalogue of recent queries and their respective data stored in the semantic cache which is basically the first key idea in semantic caching.

When a user starts a request, the Query analyzer figures out if the requested data is fully stored in the semantic cache (cache hit), partially in the cache or completely missing in the cache (cache miss). The Query analyzer also checks if the Query engine implemented in the FPGA integrates the query operations to accelerate.

If we have a cache miss or if the query engine cannot process the query (grey arrow in figure 2), the query is processed by the Spark query engine. This is the standard, non-accelerated Spark process based on libraries capable of doing complex queries. Spark fetches the data requested in HDFS and executes the query. The data and the results of the query are then sent to the query analyzer. Afterwards, the query analyzer sends the result to the user and proof check if the data and the results can fit in the cache. If they can fit in the cache, the query analyser sends them to the FPGA. When the cache update is done, the FPGA sends cache details to the query analyzer to update the cache catalogue: address of the last data sent and validity of the data cached.

If we have a cache hit and the query engine can process the query (black arrow in figure 2), the query is sent to the FPGA. The query engine executes the request and sends the result to the user and the query analyzer. Finally, if the data requested is partially in the cache, the query is parallelized into two or more queries to be executed by Spark and the query engine in FPGA. For example, referring to the example given section 2.3, the query Q1 is divided into a probe query and remainder query. The remainder query is executed by the Spark SQL and the probe query is sent to the FPGA to fetch the data from the semantic cache. Then all the results are sent to the user and remainder query's result are updated in the cache.

3.2 FPGA structure

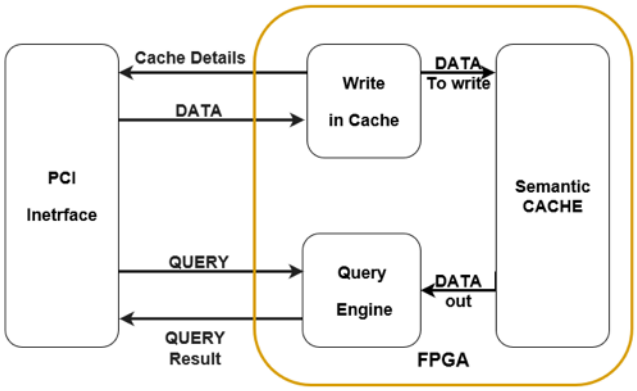

Fig. 3. FPGA structure

The structure in the FPGA (figure 3) is composed by three main components: The "Write in Cache", the "Semantic cache" and the "Query engine".

The write in cache is responsible of writing the data to the cache according to the replacement policy chosen. Furthermore, it provides the cache changes to the catalogue stored in the query analyzer. Finally, it is also responsible for managing the semantic regions.

The semantic cache memory is a combination of BRAMs linked together to constitute a beta version of a cache memory. In some large FPGAs, the total storage capacity of the BRAMS combined is up to 30 MB. In the future, we plane to extend the cache memory plan to external RAMs to create a second level of functional cache memory.

The query engine reads the query, fetches the necessary data from the cache, executes the query and sends the results to the query analyzer. The query engine must be capable of executing as much different queries as possible, reading and storing different formats of databases which is a real challenge for FPGA acceleration.

4 Experiments

In this section, we introduce the implementation details of the FPGA prototype. Then, we provide some performance results of the prototype and compare them to non accelerated standard Spark SQL.

4.1 Implementation

The prototype is composed by a server for software processing and an FPGA board for hardware processing. The intercommunication between both is managed by RIFFA [11]. RIFFA is a simple framework for communicating data from a host Central Processing Unit (CPU) to a FPGA via a PCIe. RIFFA communicates data using direct memory access (DMA) transfers and interrupt signalling. This achieves high bandwidth over the PCIe. Our implementation relies on an XpressV7-LP HE design board based on a Xilinx Virtex-7 FPGA XC7VX690T. The server used for the evaluation is based on an Intel R Xeon R CPU E5-1620 able to run at 3.60GHz with 16GB of RAM. The operating system is a Linux Ubuntu 16.04.4 Long Term Support distribution based on kernel 4.4.0-119-low-latency.

Resource	Utilisation	Available	Utilisation %
LUT	21825	433200	5.04
LUTRAM	322	174200	0.18
FF	26741	866400	3.09
BRAM	186.5	1470	12.69
IO	7	600	1.17
GT	8	20	40
BUFG	4	32	12.5
MMCM	1	20	5
PCIe	1	3	33.33

Table 1. FPGA resource utilization

Table 1 shows that the prototype is well optimized with limited resources footprint (5% of Look Up Tables (LUTs), 3% of Flip Flops (FFs) and 12% of Block Random Access Memories (BRAMs)).

4.2 Results

To test the prototype, we tested the upload speed and the download speed. Then we tested different queries and compared them to standard non-accelerated Spark.

To test the upload speed, we sent four times different sizes of data to the FPGA (from 16 Bytes to 384 KB) and got the average to have a more reliable result. Figure 4 presents the results on upload throughputs. The speed of upload which can be achieved is above 1 GB/s when the size of the data is more than 256 KB.

To test the download speed, we sent four times read queries to read different sizes of data from the FPGA (from 16 Bytes to 384 KB) and got the average to have a more reliable result. Figure 5 presents the results on download throughputs. The speed of download which can be achieved is above 1 GB/s when the size of the data is more than 256 KB.

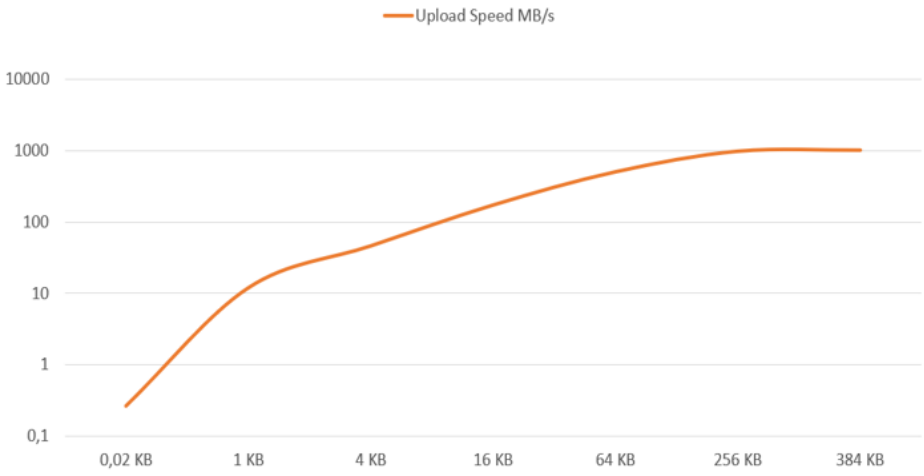

Fig. 4. Upload Speed Test

To test the query speed, we sent four times different queries for different sizes of data stored in the FPGA caches (from 16 Bytes to 384 KB) and got the average time to have a more reliable result. The different queries are:

- Simple "where" query "SELECT * FROM tableName WHERE condition where the condition is valid for all the data (returns all).
- Simple "where" query "SELECT * FROM tableName WHERE condition" where the condition is not valid for all the data (returns $\emptyset$).
- Simple "where" query "SELECT * FROM tableName WHERE condition1 and condition 2 and condition 3 and condition 4" where the conditions are valid for all the data (returns all).
- Simple "where" query "SELECT * FROM tableName WHERE condition1 and condition 2 and condition 3 and condition 4" where the conditions are not valid for all the data (returns $\emptyset$).

Figure 6 shows that the fastest query to execute is the simplest query with no valid condition while the slowest is the complex query which returns all data. This can be explained by two reasons. The first is that Query Analyzer takes more time to analyze a complex Query (between 170 μs and 140 μs) than a simple query (between 100 μs and 120 μs). The second reason is that the FPGA takes time to send Data back to the software layer when Data are present in the FPGA cache. The same figure also provides a range of throughput results for Data sizes above 256 KB.

Figure 7 presents the results of throughput comparison between Spark and the prototype. First, we did a comparison between the slowest queries and complex queries that returns all data requested. We can notice that the prototype is about 10 times faster on average than Spark.

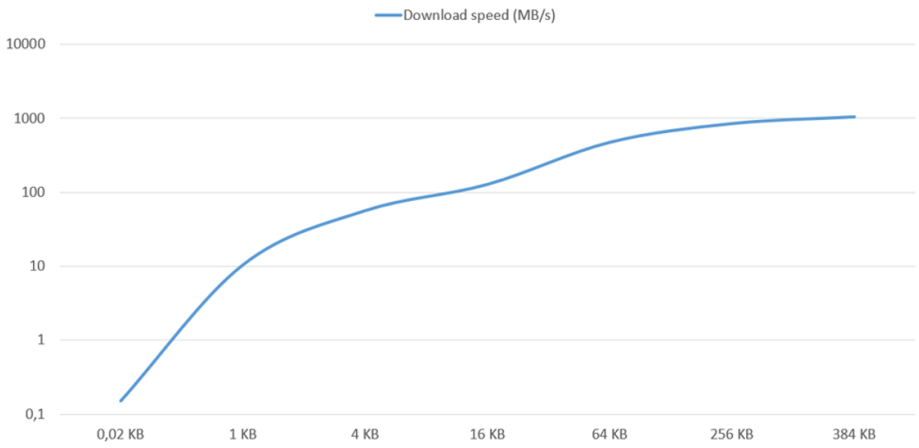

Fig. 5. Download Speed Test

5 Conclusion

In this paper, we proposed an FPGA-based architecture for the acceleration of Big Databases SQL querying. The hardware section is composed of FPGA based query engine and semantic caching. While the software section is composed of a Query analyzer, Spark SQL and the Hadoop Distributed File System (HDFS). This architecture could be later improved by introducing partial reconfiguration to the FPGA architecture to have the ability to do as much queries as possible by the query engine like what was done in [8], [18]. This architecture could also be improved by adding more than one FPGA in the architecture to have different FPGAs executing different queries at the same time. In addition, future works include extending the proposed architecture to increase parallelism and distribution. In particular two cases will be considered: (1) edge computing and (2) cloud federations.

References

1. T. R. S. Abdul Ghaffar Shoro. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 2015.
2. M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark {SQL:} Relational Data Processing in Spark. In *SIGMOD International Conference on Management of Data*, pages 1383–1394, Melbourne, Victoria, Australia, 2015.
3. A. Bansod. Efficient big data analysis with apache spark in hdfs. *International Journal of Engineering and Advanced Technology (IJEAT)*, 4(6):313–316, 2015.
4. A. Becher, D. Ziener, K. Meyer-Wegener, and J. Teich. A co-design approach for accelerated SQL query processing via fpga-based data filtering. In *International Conference on Field Programmable Technology (FPT)*, pages 192–195, Queenstown, New Zealand, 2015.

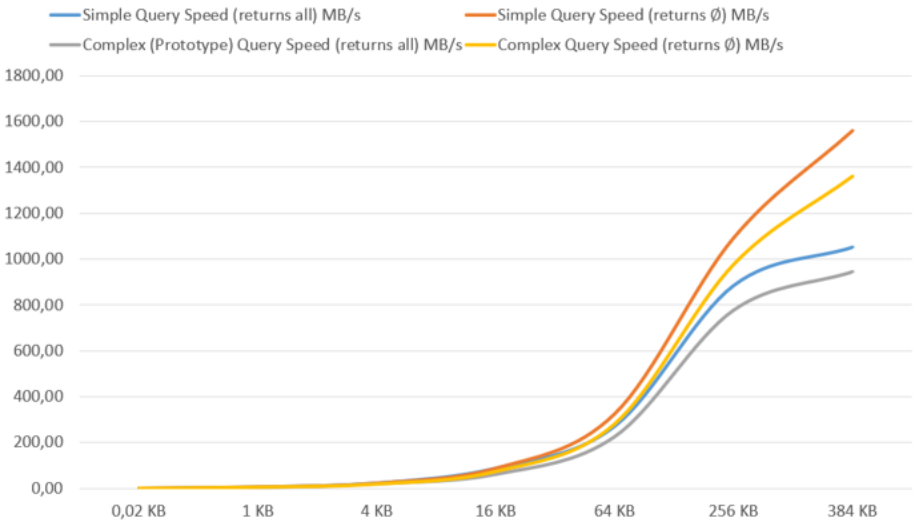

Fig. 6. Different Queries Speed Test

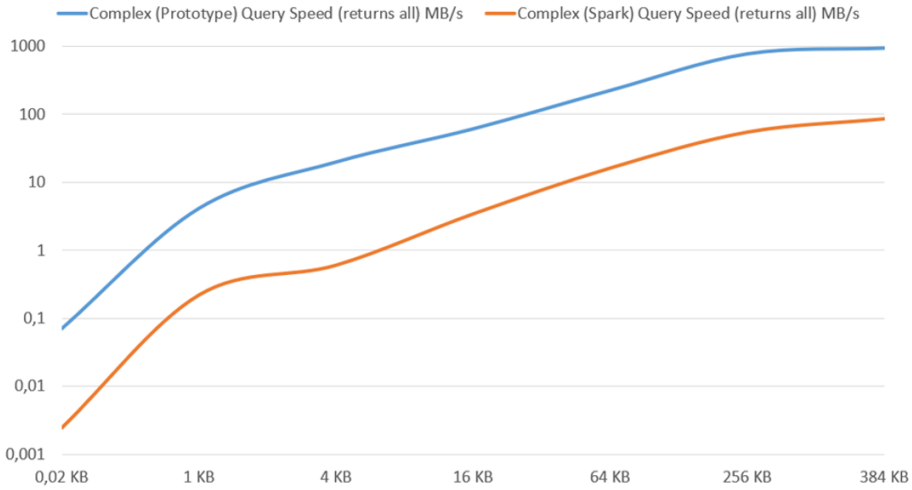

Fig. 7. Complex Queries Speed Comparison

5. M. Chen, S. Mao, and Y. Liu. Big data: A survey. *Mobile Networks and Applications (MONET)*, 19(2):171–209, 2014.
6. C. G. Cloud. Cisco global cloud index: Forecast and methodology, 2016–2021 white paper. Technical report, Cisco, 2010.

7. S. Dar, M. J. Franklin, B. b. Jónsson, D. Srivastava, and M. Tan. Semantic Data Caching and Replacement. In *International Conference on Very Large Data Bases (VLDB)*, pages 330–341, Mumbai (Bombay), India, 1996.
8. C. Dennl, D. Ziener, and J. Teich. On-the-fly composition of fpga-based SQL query accelerators using a partially reconfigurable module library. In *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52, Toronto, Ontario, Canada, 2012.
9. H. Esmaeilzadeh, E. R. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. *IEEE Micro*, 32(3):122–134, 2012.
10. S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Symposium on Operating Systems Principles (SOSP)*, pages 29–43, Bolton Landing, NY, USA, 2003.
11. M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner. Riffa 2.1: A reusable integration framework for fpga accelerators. *ACM Trans. Reconfigurable Technol. Syst.*, 8(4):22:1–22:23, 2015.
12. S. G. Manikandan and S. Ravi. Big data analysis using apache hadoop. In *International Conference on IT Convergence and Security (ICITCS)*, Beijing, China, 2014.
13. P. E. Ross. Why cpu frequency stalled. *IEEE Spectrum*, 45(4):72, 2008.
14. D. Sidler, Z. István, M. Owaida, K. Kara, and G. Alonso. doppiodb: A hardware accelerated database. In *International Conference on Management of Data, SIGMOD Conference 2017*, pages 1659–1662, Chicago, IL, USA, 2017.
15. J. Teubner. Fpgas for data processing: Current state. *Information Technology (IT)*, 59(3):125, 2017.
16. T. N. Theis and H. P. Wong. The end of moore's law: A new beginning for information technology. *Computing in Science and Engineering*, 19(2):41–50, 2017.
17. A. Vancea and B. Stiller. Coopsc: A cooperative database caching architecture. In *International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE) 2010*, pages 223–228, Larissa, Greece, 2010.
18. D. Ziener, F. Bauer, A. Becher, C. Dennl, K. Meyer-Wegener, U. Schürfeld, J. Teich, J. Vogt, and H. Weber. Fpga-based dynamically reconfigurable SQL query processing. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 9(4):25:1–25:24, 2016.